

Efficient Peer-to-Peer Information Sharing over Mobile Ad Hoc Networks

Mei Li Wang-Chien Lee Anand Sivasubramaniam
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802
E-Mail: {meli, wlee, anand}@cse.psu.edu

Abstract

Peer-to-peer systems have gained a lot of attention as information sharing systems for the wide-spread exchange of resources and voluminous information among thousands of users. However, current peer-to-peer information sharing systems work mostly on wired networks. With the growing number of communication-equipped mobile devices that can self-organize into infrastructure-less communication platform, namely mobile ad hoc networks (MANETs), peer-to-peer information sharing over MANETs is a promising research area. In this paper, we propose a distributed index structure, Multi-level Peer Index (MPI), that enables efficient peer-to-peer information sharing over MANETs. Preliminary evaluation demonstrates that MPI is scalable, efficient, and adaptive to node mobility.

1 Introduction

Peer-to-peer (P2P) systems have received a lot of attention as distributed information sharing systems for the wide-spread exchange of resources and voluminous information among thousands of users. An important research issue in P2P system is searching for resources (e.g., data, files, and services) available at one or more of these numerous host nodes. The importance of P2P searches has motivated several proposals for performing these operations efficiently, such as distributed hash table (DHT) overlay networks (e.g., CAN [9], CHORD [12]).

However, these P2P information sharing systems only work on wired networks. Rapid advance in wireless technology along with greater demands for user mobility motivates recent intensive research efforts in mobile ad hoc networks (MANETs). MANETs consist of mobile devices equipped with interfaces for short to medium range wireless communications, such as laptop, Personal Digital Assistance (PDA), etc., that can spontaneously self-organize into a communication structure without requiring any fixed in-

frastructure. Most of the recent research efforts in MANETs are focused on the lower layers, such as link layer, network layer and transport layer, to enable efficient communication among nodes in the network (a survey is provided in [4]). On the other hand, it is envisioned that the future gadgets such as MP3 players and DVD players will be equipped for wireless communications via technologies such as Wi-Fi and Bluetooth. Thus, MANETs will enable future applications such as sharing and exchange of documents, pictures, music, and movies on mobile devices. These kinds of applications resemble information sharing in P2P systems in terms of lack of centralized servers and equal roles taken by nodes in the system, thereby we call them *mobile peer-to-peer information sharing (MP2PIS)*.

Similar to P2P systems, to enable effective data exchange in MP2PIS, a first step is to devise search mechanisms to find a data object of interest in MANETs. However, due to the unique characteristics of MANETs compared to P2P systems, such as limited transmission range, resource (CPU, bandwidth, storage) constraints, and node mobility, simply importing DHTs to MANETs is not a valid solution. In this paper, we propose a distributed index structure, *Multi-level Peer Index (MPI)*, that enables efficient P2P information search over MANETs. Through simulation, we demonstrate that MPI is scalable, efficient and adaptive to node mobility.

The remaining part of this paper is organized as follows. After introducing the system model and motivations behind our proposal in next section, we present MPI in Section 3. Performance evaluation is presented in Section 4. We compare our proposal with related works in Section 5. Conclusion remarks and future directions are given in Section 6.

2 Preliminaries

In this section, we first introduce the system model that our proposal is based upon. Then we outline the motivations behind our proposal.

2.1 System Model

Each node in MANETs has limited radio range and we assume that all nodes have the same radio range. A node can communicate with other nodes within its radio range directly or the ones out of its radio range indirectly through multi-hop relay. Any node may join or leave the network, resulting in dynamic membership changes. In addition, nodes may move, resulting in dynamic topology changes.

Each node is assumed to know its own position as well as its immediate neighbors' position through GPS devices or other means. We use Greedy Perimeter Stateless Routing (GPSR) [5], a well-known geographical routing protocol for wireless networks, as our basic routing protocol (necessary modifications are detailed in the later part). GPSR consists of two methods for forwarding packets: greedy forwarding that forwards a packet progressively closer to the destination, and perimeter forwarding that forwards packets out of a void region where greedy forwarding fails.

Besides forming a communication structure, each node also provides certain number of sharable data objects. We assume that each data object is associated with a well-known single-attribute key. The search mechanism is to find the node storing the data object with specified key and obtain the relevant data object thereafter. In this paper, we focus on the case when a user is interested in obtaining one arbitrary data object (instead of all data objects) satisfying the specified search criteria if there are multiple such data objects.

Following terms will be used in the rest of this paper. *Source node* of a data object is defined as the node storing this data object. *Index information (or index entry)* of a data object is the mapping between the data object and the NodeID of its source node, i.e., the pair $\langle \text{key}, \text{NodeID} \rangle$, while *index node* of a data object is defined as the node storing the index information for this data object. *Location information (or location entry)* of a node is the mapping between the NodeID of this node and its physical location, i.e., the pair $\langle \text{NodeID}, \text{location} \rangle$, while *location node* is defined as the node storing the location information for this node.

2.2 Motivations

Due to potential large scale, scarce resources and constant node movement, an efficient search mechanism for MP2PIS should satisfy following requirements.

- **Scalability.** A search mechanism should be scalable in terms of network size as well as the amount of sharable information stored in the network.
- **Efficiency.** Due to the scarce resources, it is desirable that a node can find a nearby source node without travelling much further than the source node itself.

- **Adaptivity to node movement.** The index needs to be distributed intelligently amongst nodes so that the index structure can adapt to node movement without incurring high update overhead.

3. Multi-level Peer Index

To achieve the goals as described in previous section, namely, scalability, efficiency and adaptivity to node movement, we propose a distributed index structure, Multi-level Peer Index (MPI).

3.1 Overview

To address scalability, the basic structure of MPI is built upon well-known hashing techniques so that each node has relatively equal share of index entries and small storage requirement. In MPI, data objects are hashed into geographical coordinates within the network region (we assume that the network boundary is known). Therefore, the physical network region becomes the virtual search space, which is partitioned into multiple sub-spaces and then assigned to nodes.

To achieve efficiency, any requester ideally should find the nearest source node storing data objects of interest without traversing much further than the physical distance between the requester and the source node. With this design goal in mind, we embed hierarchical spatial information in the index so that nearby nodes can take advantage of their physical proximity.

We now discuss how to improve the system's adaptivity to dynamics. The network is virtually partitioned into grid cells where nodes in a grid cell are collectively responsible for the assigned keys. With this design, as long as a node moves within its grid cell, there is no need for update, incurring much lower update overheads compared to GHT [10] which assigns keys to a specific node residing at a specific geographical coordinate.

To deal with node mobility, we embed a location lookup service, namely *Multi-level Location Service (MLS)*, in the multi-level structure of MPI. At the same time we decouple index and location entries. The benefits for this embedding and decoupling are three-folds. First, the location service can take advantage of the nice properties of this multi-level structure as well. Secondly, with this embedding, we have a single protocol that provides both data lookup as well as location lookup. Lastly, decoupling index entries and location entries renders the flexibility to update a node's location and index separately.

Using MPI, search consists of three phases: data lookup, location lookup and data retrieval. In the data lookup phase, MPI is used to find the NodeID of a nearby source node storing data objects of interest. The location lookup phase

is then conducted to obtain the location information of the source node and subsequently reach the source node with the assistance of MLS. Requested data object is then obtained from the source node.

3.2 Index Structure of MPI

To embed nested spatial information in the index structure, we partition the network hierarchically as follows. The entire region is partitioned into m equal-sized squares while each of these squares is partitioned further into m smaller children squares and so on. We label the squares from the highest level (associated with the entire region) to the lowest level (associated with the square with minimum size) with increasing numeric value, i.e., the highest level has label 1. Every node in each of these squares collectively constructs a hash index (the details will be explained shortly), forming a multi-level index structure. With this design, MPI does not assign higher responsibilities to any node, avoiding performance bottleneck and single point of failure which are normally associated with hierarchical tree structures.

We now present how nodes within a square collectively construct a hash index. The whole network is associated with a *primary hash function*, which takes the key value of a data object and the geographical boundary of a square as inputs, and generates an output that is a geographical coordinate bounded by the specified boundary. When a node joins the network, it publishes the index information of its locally stored data objects as follows. For each of its locally stored data objects, it first calculates a set of hashed geographical coordinates using the geographical boundaries associated with each of the squares that it resides in and the key of the data object as inputs. Then it publishes the index information for the data object to these geographical coordinates (how to choose index nodes around these coordinates will be detailed shortly).

Through MPI, the requested data objects can always be found within the smallest square where both the requester and the source node reside in. This search efficiency comes with the overhead of publishing index information to each level of MPI. We expect that the number of levels in MPI is small, and search request rate will be very high compared to the node join rate, thereby this overhead is expected to be reasonable.

To implement the grid structure, in the lowest level (minimum sized) squares, the spatial region is further partitioned into regular-sized grid cells, and all nodes within a grid cell become the index nodes for data objects hashed into the grid cell. The downside of the grid approach is that the index information needs to be propagated to all nodes within a grid cell. To make this overhead reasonable, the size of a grid cell should not be too large. In our current design, we set the side length of a grid cell, L , to $\frac{r}{\sqrt{2}}$ where r indicates the

radio range so that any two nodes in a grid cell can communicate directly, i.e., they are within each other's radio range.

In some rare cases, it is still possible that there is no nodes existing in a grid cell temporarily due to node movement. To deal with this issue, we apply a *secondary hash function* to choose *surrogate index nodes* (or *surrogate location nodes*) to store index entries (or location entries) that are originally assigned to an empty grid cell (using the primary hash function).

Figure 1(a) gives an example for MPI with 4-level hierarchy when $m = 4$. The first level (entire region) denoted by Q is partitioned into four level-2 squares Q_1, Q_2, Q_3, Q_4 (we name the four squares in clockwise order starting from the one at top left corner). Then each of these squares is partitioned into four level-3 squares. For instance, Q_1 is partitioned into $Q_{11}, Q_{12}, Q_{13}, Q_{14}$. Node 1 in Q_{111} has a data object with Key A and it publishes the index information for this data object to the entire region Q and the three squares Q_1, Q_{11} , and Q_{111} that it resides in. Here the grey circles, $I_{1,A}, I_{2,A}, I_{3,A}$, and $I_{4,A}$ represent the index nodes for Key A in corresponding regions.

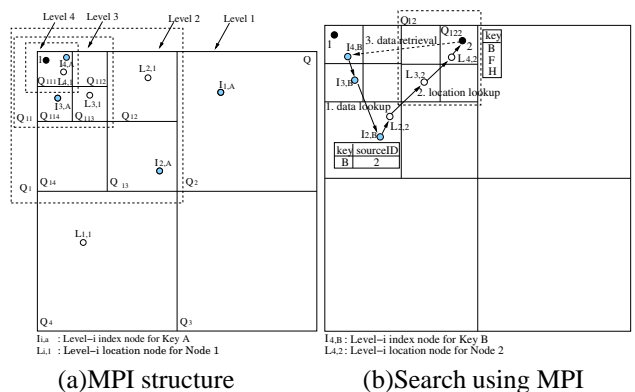


Figure 1. An illustrative example for MPI.

3.3 MLS

As discussed in Section 3.1, to address node mobility, we embed the location lookup structure, Multi-level Lookup Service (MLS), naturally in the multi-level structure of MPI and at the same time decouple index entries and location entries. Specifically, a node chooses its location nodes at each level by applying the same hash function and grid structure of MPI, except here NodeID instead of key value for a data object is used as one input for the hash function.

To reduce location update overhead, we hide certain degree of movement from higher levels of the hierarchy so that location nodes at different levels can react differently to node mobility at different granularities. In MLS, nodes at lower level squares maintain finer location information

while nodes in higher level squares only maintain coarser location information. Location nodes at the lowest level has detailed location information while the ones at higher level (i.e., level- i) square only maintain a pointer pointing to one of its children squares (i.e., level- $(i+1)$) square) that a node resides in.

Figure 1 also gives an example for MLS. Node 1 publishes its location information to Q and the three square Q_1 , Q_{11} , and Q_{111} . Here the white circles, $L_{1,1}$, $L_{2,1}$, $L_{3,1}$, and $L_{4,1}$ represent the location nodes for Node 1 in these corresponding regions.

After explaining the general idea of MLS, we discuss how MLS structure is updated upon node movements. When a node moves from its previous grid cell x to a new grid cell y , it becomes one of the location nodes for those nodes that have hashed coordinates mapped to its current grid cell y and is no longer a location node for those nodes having hashed coordinates mapped to grid cell x . Therefore, it deletes its old location information stored locally and obtains the location information from any node in grid cell y . In addition, when a node moves from its previous level- i square to a new level- i square, it publishes its new location to the location nodes in its current level- i square, and deletes the location information from the location nodes in the old level- i square. Furthermore, it updates the location information at the location nodes in its parent level- $(i-1)$ square so that location nodes in this square point to the level- i square that this node is currently residing in.

3.4 Search

When a node issues a search request, it includes the key associated with the requested data object, its NodeID and current location. We discuss the three phases in search, namely data lookup, location lookup and data retrieval, respectively, in this section.

Data lookup: When a node issues a search request, it first check its own local data store. If a result is found, the search terminates. Otherwise, it invokes data lookup by sending the request to an index node (i.e., the first index node reached in the relevant grid cell) in its lowest level (i.e., level- H where H is the maximum level of MPI) square¹. If the index node does not have an entry for the requested key, the request is then forwarded to the index node at next higher level square (i.e., the parent square). The process is repeated until either the index entry for the requested key is found or the top level of MPI is reached without finding an index entry for the requested key. For the former case, the location lookup as described below is invoked. For the latter case, the search fails and a failure

¹For index/location publish and data/location lookup, a message's destination is a grid cell instead of a point as in GPSR. GPSR is modified with the centroid of the grid cell as the pseudo destination of these messages.

message is returned back to the requester node.

Location lookup: The location lookup is invoked at the square where the index entry for the requested key is found. Assuming that this square is a level- x square. The location request is first routed to the location node responsible for the NodeID of the source node at this square. From this location node, the requester will either obtain a pointer pointing to a level- $(x+1)$ square (i.e., one of its children squares) that the source node resides in or the precise location information of the source node (if level- x is the lowest level). In the former case, the request is forwarded to the location node in the pointed level- $(x+1)$ square. This process is repeated until the lowest level square (i.e., level- H) is reached where the precise location of the source node is found. At this point, data retrieval as described below is invoked.

Data retrieval: Since the location of the requester node is included in the request message, data retrieval can be done easily if the requester node stays in its original grid cell. However, there are two possible cases when a node moves out of its original grid cells. The first case is that a node moves out of its original grid cell but still within its lowest level (i.e., level- H) square. In this case, when the reply message reaches its original grid cell, a location lookup is invoked in this square and the reply message is forwarded to the new location of the requester. The second case is that the requester moves out of its level- H square. To handle this case, a node always leaves a forwarding pointer at its old location nodes which points to the grid cell that it moves into. When the reply message reaches the old square specified in the reply message, the forwarding pointer is obtained from the location nodes so that the reply message can be delivered to the current square that the node resides in. The reply message might go through multiple forwarding pointers before it reaches the current location of the requester.

Figure 1 (b) gives a search example in 4-level MPI while Node 1 searches for Key B. In this example, Node 2 stores the data object with Key B. Node 1 first invokes data lookup by forwarding the search message to Key B's index node in its level-4 square, $I_{4,B}$. Assume that this index node does not has an index entry for Key B. The search message is then forwarded to level-3 index node, $I_{3,B}$. This process is repeated until the search message reaches level-2 index node, $I_{2,B}$, which has an index entry indicating that Node 2 has the requested data object. At this point, location lookup for Node 2 is invoked in this level-2 square, Q_1 . The location lookup request is forwarded sequentially to $L_{2,2}$, $L_{3,2}$ and $L_{4,2}$, which are the location nodes at level-2, level-3 and level-4 squares, respectively. After the location for Node 2 is obtained from $L_{4,2}$, the message is then forwarded to Node 2 and data retrieval is invoked thereafter.

3.5 Index maintenance

When a node joins the network, it needs to perform two tasks: publishing its data objects and location information (which has been explained previously); obtaining index information and location information hashed to the grid cell that it resides in. To obtain these index and location information, upon joining the network, a node broadcasts a HELLO message containing its NodeID. Any node within its grid cell can hear the HELLO message since they are within the radio range. After hearing the HELLO message, a node waits for a random interval before it replies with its current index entries as well as location entries. One node will reply first and any other nodes in the grid cell can hear this reply and will not reply again.

When a node moves out of its previous grid cell, it becomes one of the index nodes for those data objects that have hashed coordinates mapped to the new grid cell that it moves into and is no longer an index node for those data objects having hashed coordinates mapped to its previous grid cell. Therefore, it deletes its old index information and obtains the index information from any node in its current grid cell. In addition, a node needs to republish the index information for its data objects when it moves out of a level- i square and enters a new level- i square.

4. Simulation evaluation

In this section, we first present our simulation setup, following which we present the preliminary simulation results.

4.1 Simulation setup

We implement a simulator using csim[11]. The network setup, workload and performance metrics are as follows:

Network setup: In the simulation, each node has a radio range 250 meter. The network sizes are 64, 256, 1024, and 4096. The default network size is set to 1024 if unspecified otherwise. m is set to 4 and the lowest level square of MPI is set to contain 4 grids. The nodes are initially randomly placed in a square region, whose area is scaled with the number of nodes so that the average density is 4 nodes in a 175*175 square region. All nodes move using the random waypoint model [1] with a maximum velocity ranging from 0m/s to 20m/s. The pause time is set to 0 second.

Workload: Each node holds 10 data objects. A node issues random searches into the network while the average time interval between two searches issued by a node is 20 seconds. The simulation time is 500 seconds. The results shown in following sections are averaged over 10 trials for each set of the simulation.

Performance metrics: **Path length** is the average number of hops traversed from requester node to source node

in a search (including the data lookup and location lookup phases). This metric indicates search latency. **Path stretch** is defined as $\frac{real_path_length}{ideal_path_length}$ where $real_path_length$ is defined as above and $ideal_path_length$ is the number of hops along the shortest path between the requester node and the source node. This metric indicates how close the path taken in a search is to the shortest path between the requester node and source node. **Message number** is the total number of messages, including search messages, index publish/update messages, location publish/update messages, and other control messages such as HELLO messages, processed by a node per second.

4.2 Results

In this section, we first show the effect of network size. The effect of nodes' moving speed is then followed. We use the basic flooding approach as the baseline for comparison.

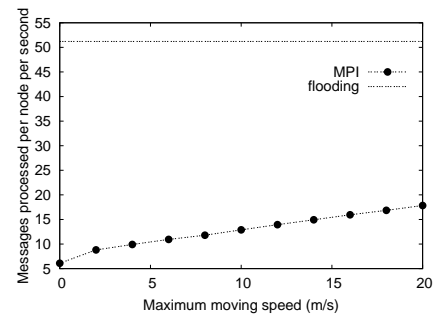


Figure 3. Effect of moving speed on message number.

Effect of network size. Figure 2 shows the path length, path stretch, and message number when we vary the network size from 64 to 4096. From Figure 2(a), we can see that the path length increases slowly with the network size. The path stretch shown in Figure 2(b) is bounded by a small constant number (i.e., 5). Figure 2(c) shows the message number. We include the message overhead for flooding technique here for comparison. From this figure, we can see that the message number of flooding is much higher than the message number of MPI. These results confirm our expectation that MPI is scalable and efficient.

Effect of node mobility. The relationship between message number and moving speed is illustrated in Figure 3. The message number increases linearly with the maximum moving speed. However, even at the highest moving speed 20m/s, the message number is still much lower than flooding technique. These results demonstrate the adaptivity of MPI to node mobility.

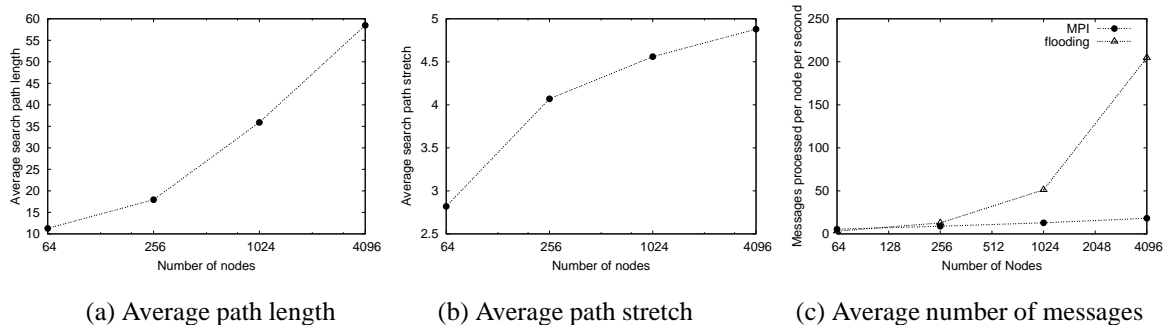


Figure 2. Effect of network sizes.

5. Related Work

A few studies address various issues in peer-to-peer information sharing over MANETs. In ORION [7] and 7DS [8], the authors apply cooperative caching concept to enable peer-to-peer file sharing among nodes in MANETs. However, using their techniques, the success rate of a search is not predictable and highly depends on the search locality in the system. [2] proposes a cross-layer protocol that embeds a search mechanism in a geographical location service. This coupling of index information with the location information incurs high overhead when a mobile node stores multiple data objects since it requires that the index information for each of the data objects be updated whenever the node moves. In [6], a P2P platform for developing mobile P2P applications is proposed. [3] addresses the data dissemination issue in ad hoc networks by partitioning a file into multiple segments so that a node can acquire different segments at different times and in different locations, improving the data retrieval efficiency. However, the issue of how to discover a particular file of interest is not addressed in this paper.

6. Conclusion

Enabling efficient peer-to-peer information sharing over MANETs (MP2PIS) is foreseen to be a promising application and vital research area. One of the challenges for MP2PIS is to develop efficient search mechanism that can find requested data object quickly in an environment with great node mobility and scarce resources. In this paper, we propose a distributed index structure, *Multi-level Peer Index (MPI)*, which consists of data lookup service and location lookup service, to enable efficient search in MANETs. Preliminary evaluation demonstrates that our proposal is scalable, efficient and adaptive to node mobility.

We are conducting more in-depth analysis and simulation to evaluate the effect of various system parameters, such as size of grid cell and number of levels in MPI, on the performance of MPI. We are also investigating how to

expand the search ability of MPI to more complex query types, such as range queries, multi-attribute queries, etc..

References

- [1] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of MOBICOM*, pages 85–97, October 1998.
- [2] G. Ding and B. Bhargava. Peer-to-peer file-sharing over mobile ad hoc networks. In *Proceedings of IEEE Annual Conference on Pervasive Computing and Communications Workshops*, pages 104–109, March 2004.
- [3] S. K. Goel, M. Singh, D. Xu, and B. Li. Efficient peer-to-peer data dissemination in mobile ad-hoc networks. In *Proceedings of International Conference on Parallel Processing Workshops*, pages 152–158, August 2002.
- [4] X. Hong, K. Xu, and M. Gerla. Scalable routing protocols for mobile Ad hoc networks. *IEEE Network Magazine*, pages 11–21, July/August 2002.
- [5] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of MOBICOM*, pages 243–254, August 2000.
- [6] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In *Proceedings of Peer-to-Peer Computing (P2P)*, pages 75–93, August 2001.
- [7] C. Lindemann and O. P. Waldhorst. A distributed search service for peer-to-peer file sharing in mobile applications. In *Proceedings of International Conference on Peer-to-Peer Computing (P2P)*, pages 73–80, September 2002.
- [8] M. Papadopouli and H. Schulzrinne. Effects of power conservation, wireless coverage and cooperation on data dissemination among mobile devices. In *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, pages 117–127, October 2001.
- [9] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of SIGCOMM*, pages 161–172, August 2001.
- [10] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu. Data-centric storage in sensor networks with GHT, a geographic hash table. *Mobile Networks and Applications (MONET), Special Issue on Mobile and Wireless Data Management*, 8(4):427–442, August 2003.
- [11] H. Schwetman. Csim18 - the simulation engine. In *Proceedings of the 1996 Winter Simulation Conference*, pages 517–521, December 1996.
- [12] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of SIGCOMM*, pages 149–160, August 2001.